A Formal Approach to Negotiating Agents Development*

Marlon Dumas[1], Guido Governatori[2], Arthur H.M. ter Hofstede[1], Phillipa Oaks[1]

[1] Centre for Information Technology Innovation
Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{m.dumas, a.terhofstede, p.oaks}@qut.edu.au

[2] School of ITEE
The University of Queensland
Brisbane QLD 4072, Australia
guido@itee.uq.edu.au

Abstract

This paper presents a formal and executable approach to capture the behaviour of parties involved in a negotiation. A party is modeled as a negotiating agent composed of a communication module, a control module, a reasoning module, and a knowledge base. The control module is expressed as a *statechart*, and the reasoning module as a *defeasible logic* program. A strategy specification therefore consists of a statechart, a set of defeasible rules, and a set of initial facts. Such a specification can be dynamically plugged into an agent shell incorporating a statechart interpreter and a defeasible logic inference engine, in order to yield an agent capable of participating in a given type of negotiations. The choice of statecharts and defeasible logic with respect to other formalisms is justified against a set of desirable criteria, and their suitability is illustrated through concrete examples of bidding and multi-lateral bargaining scenarios.

Keywords. Automated Negotiation, Software Agent, Defeasible Logic, Statecharts.

1 Introduction

As the amount of commercial transactions carried out through the Internet increases at a spectacular rate, the interest for partially or totally automating the negotiation of the terms of these transactions has rapidly become a hot research topic. Consequently, automated negotiation has evolved in a few years from a futuristic vision to a promising technology [24]. In particular, optimal strategies for several forms of automated bargaining and auctioning under simplifying assumptions have been identified, and the extensions of these results to more realistic settings have been studied by the game-theory and the distributed artificial

*This work was supported by the Australian Research Council SPIRT Grant "Self-describing transactions operating in a large, open, heterogeneous, distributed environment" involving QUT and GBST Holdings Pty Ltd.

intelligence communities [30, 32, 22]. Furthermore, several tournaments where automated traders compete to maximize their profits in electronic auction houses have been organized (e.g. [33, 20]), and their results are quite encouraging.

Software agents are becoming a choice technology for carrying out automated negotiations [32]. In this approach, each party is represented through an agent which interacts either directly with the other parties, or through a broker. The behaviour of the agent is guided by a *strategy*, which may be as simple as continuously increasing the current offer by the minimum increment up to a maximum amount (e.g. in the context of an English auction or a bargaining scenario), or as complex as determining the next offer based on an analysis of the history of all the previous offers and even that of previous negotiations [14, 9]. The issue in this context is thus "how to express a negotiation strategy?".

An approach developed at MIT's Kasbah project [7] in the setting of English auctions, is to provide a set of simple pre-defined strategies, corresponding to linear, quadratic and exponential functions over time. The authors argue that using pre-defined strategies has the advantage that it makes the bidding agents' decisions easily explainable and predictable. However, this approach considerably restricts the possibilities of the users. In particular, it is not applicable when the users' valuation of the auctioned item depends upon that of the other bidders (i.e. in *correlated-value auctions* [32]), since in this case, the bidding function should dynamically adapt to the other participants' bids. In addition, the use of this approach in a broader setting than English auctions, implies that there will be one set of alternative strategies for each negotiation protocol (i.e. one set of agents for each variant of the double auction, another for each kind of bargaining, etc.).

At the other extreme, each participant involved in a negotiation could develop its own negotiating agent "from scratch", using a general-purpose design methodology and development environment (see [29] for a list of agent building tools). Although this approach is unavoidable when very complex strategies are considered, its systematic use leads to time-consuming and hard-to-reuse development efforts. In addition, agents developed under this approach have static functionalities, in the sense that it is often not possible to modify their behaviour without having to rebuild them from scratch.

As an alternative to these extreme approaches, our aim is to develop a simple yet expressive framework for specifying negotiating agents' strategies, in a way that their decisions are predictable and explainable. Specifically, we explore the suitability of defeasible logic programming [5] for expressing the decision-making process of negotiating agents, coupled with statecharts [18] for expressing their internal coordination. We argue that defeasible logic is suitable for expressing negotiation strategies, since it straightforwardly captures concepts such as preferences, hypotheses, arguments and counter-arguments. On the other hand, statecharts provide a comprehensive and concise approach to express the possible states and activities of a negotiating agent.

An important feature of the above approach is that it leads to plug-and-play negotiation strategy specifications. Indeed, given an agent shell capable of interpreting statecharts and defeasible logic programs, it is possible to dynamically plug a strategy specification into this shell, so as to obtain a negotiating agent.

To validate the viability of our approach, we have applied it to several case-studies. In

this paper, we develop two of them in details: one concerning an English auction, and the other one concerning a one-to-many bargaining scenario, where one buyer simultaneously bargains with several potential sellers in order to find an acceptable deal. This validation work follows the preliminary results presented in [15], where we show how defeasible logic can be used to express strategies for brokered trading, and for a simple case of bargaining.

The paper is structured as follows. In section 2, we introduce our approach and justify our choices against a set of design criteria. Next, in section 3, we develop two case-studies respectively involving an English auction and a bargaining scenario. Finally, in section 4 we compare our proposal to related ones, before concluding in section 5.

2 Rationale, approach, and enabling formalisms

2.1 Desiderata

Before choosing one or several languages for the specification of negotiation strategies, it is important to establish a set of criteria that such languages need to satisfy. The criteria presented below are inspired from those formulated by [19] in the context of techniques for information modelling. They encompass several well-known principles of language design.

Firstly, a language for specifying negotiation strategies needs to be *formal*, in the sense that its syntax and its semantics should be precisely defined. This ensures that the strategy specifications can be interpreted unambiguously (both by machines and humans) and that they are both *predictable* and *explainable*. In addition, a formal foundation is a prerequisite for verification purposes.

Secondly, the language should be *conceptual*. This, following the well-known *Conceptualization Principle* of [17], effectively means that it should allow its users to focus only and exclusively on aspects related to strategies, without having to deal with any aspects related to their realisation or implementation. Examples of conceptually irrelevant aspects in the context that we consider are: physical data organisation and access, platform heterogeneity (e.g. message-passing formats), and book-keeping (e.g. message queue management).

Thirdly, in order to ease the interpretation of strategies and to facilitate their documentation, the language should be *comprehensible*. Comprehensibility can be achieved by offering a graphical representation, by ensuring that the formal and intuitive meanings are as much in line as possible, and by offering structuring mechanisms (e.g. decomposition). These structuring mechanisms often lead to *modularity*, which in our setting means that a slight modification to a strategy should concern only a specific part of its specification. Closely related to its comprehensibility, the language that we aim for, should be *suitable*, that is, it should offer concepts close to those required in negotiations. In particular, the language should be able to address the following aspects: coordination between concurrent negotiation threads, communication with external parties, partial, incomplete and dynamic descriptions, hypothetical reasoning, defeasibility/belief revision, and argumentation.

As we are interested in the automation of the negotiation process, the strategy description language should be *executable*, and its execution should exhibit acceptable perfor-

mances even for complex strategies involving many issues (i.e. the execution performances should be *scalable*).

Finally, the language that we aim for should be sufficiently *expressive*, that is, it should be able to precisely capture a wide spectrum of strategies.

2.2 An architecture for negotiating agents

We view a negotiation process as a set of software agents which interact in order to reach an agreement. Agents participating in a negotiation can interact directly or through a broker. In some situations, the role of a party during the negotiation process is almost entirely carried out by the broker. This is the case for instance in some auction houses, where the auction broker takes the place of the seller.

Following an abstract architecture for agents with memory presented in [36], each of the software agents is composed of four modules: (i) a memory which contains the history of the past decisions and interactions of the agent, including its current intentions, (ii) a communication module responsible for receiving and sending messages to the other agents and interacting with the user, (iii) a reasoning module which encodes the decision-making part of the agent, and (iv) a control module which coordinates the other components. As a refinement to this architecture, we choose to express the control module as a statechart (see section 2.3), the reasoning module as a defeasible logic program (see section 2.4), and the memory as a knowledge base of strict and defeasible facts. The architecture can then be depicted as in figure 1.

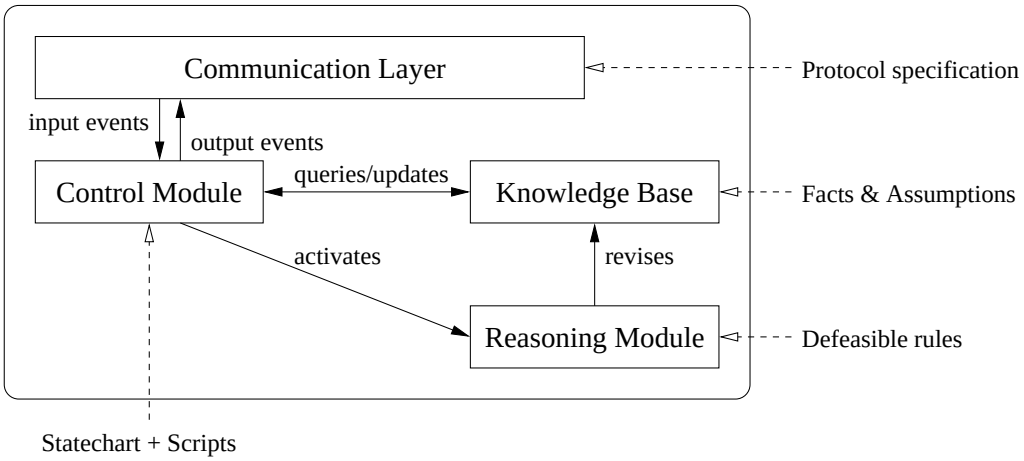

Figure 1: Architecture of a negotiating agent.

Conceptually, each time that a negotiating agent is notified of a change in the negotiation status, it updates the base facts stored in the knowledge base accordingly, and it activates the reasoning module. The reasoning module reads these facts from the knowledge base, attempts to deduce new facts and refute existing ones, and updates the derived facts stored in the knowledge base accordingly. Depending on the state of the knowledge

base after this revision process, the control module determines whether it should ask the communication module to submit a proposal or counter-proposal immediately, or it should wait for some further event, or retract from the negotiation. It is also through the communication module that the control module communicates with the agent responsible for managing the user interface (which is not shown in figure 1). This separation between the agent responsible for handling the negotiation process, and the one responsible for interacting with the user (e.g. for collecting the parameters of the negotiation, or for displaying its status), adds considerable flexibility to the architecture. In particular, these two agents can be located in different machines, and the negotiating agent can even be mobile.

To summarise, a negotiation strategy is composed of a statechart, a set of "scripts", a defeasible logic program, a set of initial facts, and the addresses of the other parties involved in the negotiation. This strategy is executed by an agent shell, composed of a statechart interpreter, a defeasible logic inference engine, and a communication layer. Therefore, a strategy can be seen as a plug-in component, which can be dynamically loaded into an agent shell to obtain a negotiating agent.

2.3 Expressing internal coordination: Statecharts

When it comes to model control-flow, automata-based notations are a natural choice, since they are conceptual, formal, executable, comprehensive, and suitable for modeling concepts such as branching and loops. Among the many automata-based notations that can be found in the literature, statecharts [18] have proven to be suitable for designing complex systems, since they provide constructs for capturing concurrency, they offer a good degree of modularity, and they provide better comprehensiveness than plain automata, by considerably reducing the number of states and transitions required in a specification.

For these reasons, statecharts are widely used in the area of reactive systems development, and are becoming a standard for intra-object coordination modeling, as they are part of the Unified Modeling Language (UML) [31]. In the sequel, we adopt UML's syntax of statecharts, which is slightly different from the one described in [18].

Statecharts are finite state machines whose states are optionally labeled by a name and a list of actions, and whose arcs are optionally labeled by Event-Condition-Action rules.

The occurrence of an event can fire a transition if (i) the machine is in the source state of the transition, (ii) the type of the event occurrence matches the event description attached to the transition, and (iii) the condition of the transition holds. An event occurrence can be the reception of a signal, a change in the system's clock, or a change in the truth value of a condition. The event, condition, and action parts of a transition are all optional. A transition without an event is called a *triggerless transition*. Triggerless transitions are enabled when the action(s) attached to their source state is (are) completed. On the other hand, transitions labeled by events can fire during the execution of the sequence of actions attached to their source state. In this case, this execution is interrupted.

A *compound state* is one whose action part is described itself as one or several statecharts. Compound states come in two flavours: OR and AND. An OR-state contains a single statechart, while an AND-state contains several statecharts which are intended to be

executed concurrently. Each of these statecharts is called a *concurrent region*. Concurrent regions are separated from each other by a dashed line. Composition is recursive, that is, the substates of a compound state may themselves be decomposed either as OR-states or as AND-states.

The statechart in figure 2 describes the control module of an agent participating in three auctions. The agent starts by contacting each of the auctioneers in parallel, asking for the current quote (see the three concurrent regions separated by dashed lines). As soon as a quote arrives, it is inserted in the knowledge base, and when the three quotes have arrived, the reasoning module is activated (see state "Auction selection"), so as to determine whether a bid should be submitted, and to which auction. If a bid is submitted, then the agent waits for an answer, which can be one of two: either another participant overbids, in which case the whole process has to be restarted, or the bid wins the auction.

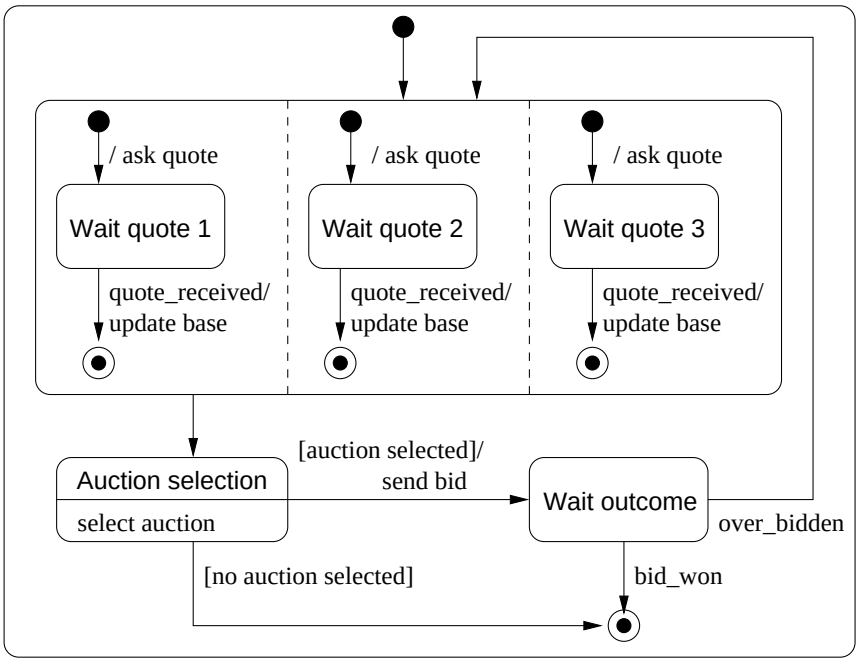

Figure 2: Extract of the control module of an agent participating in several auctions

2.4 Expressing flexible decision-making: Defeasible Logic

Applying a negotiation strategy in a particular context is an intensive decision-making process. While most aspects of this decision-making process could be fully captured in classical logic programming (which has a formal semantics and has proven to be a powerful tool for building decision-making systems), this would put a burden on the developers of strategies, since logic programming is a generic paradigm and offers nothing specific to deal with the common characteristics of negotiation strategies such as argumentation, hypothet-

ical reasoning, partial, incomplete and dynamic descriptions (see suitablity requirement in section 2.1).

It is well known that non-monotonic reasoning is appropriate to model the above characteristics [1]. Accordingly, we propose to express the decision-making aspects of negotiation strategies using a language based on non-monotonic reasoning. Non-monotonic reasoning is a complex phenomenon with many facets. Consequently, a plethora of non-monotonic logics have emerged in the past years. Unfortunately, most of them have different and sometimes incompatible intuitions, they are computationally intractable [6], and they have been used only in a few standard toy examples, whereas real-life applications require low complexity and the ability to handle more complicated cases.

In this paper, we advocate the use of Defeasible Logic (DL) [5] to model negotiation strategies. This logic offers the advantages of non-monotonic reasoning for negotiation strategy specification (such as the ability to handle partial, incomplete and dynamic descriptions), while not suffering from the drawbacks mentioned above. The following are the major reasons supporting this choice:

- A negotiation can be thought of as a dialogue between parties concerning the resolution of a dispute. This suggests that argumentation based reasoning formalisms are suitable to characterise it. In [16], it was shown that DL can be characterised by an argumentation semantics, thus the formal semantics of defeasible logic is in line with the argumentative nature of negotiations.

- Given the close connection between derivations in DL and arguments, it follows that negotiation strategies expressed in DL are explainable.

- In contrast to many other non-monotonic logics, DL is "skeptical", meaning that it does not support contradictory conclusions. Instead DL seeks to resolve conflicts. In cases where there is some support for concluding A but also support for concluding $\neg A$, DL does not conclude any of them (thus the name "skeptical"). If the support for A has priority over the support for $\neg A$ then A is concluded. It can be argued that non-skeptical reasoning is inappropriate for modeling decision-making processes such as negotiations, since it is undesirable to deduce both that a decision should be taken, and that it should not be taken.

- DL is representative of non-monotonic reasoning formalisms: many variants of DL have been proposed and it has been shown that these variants are able to capture most of the different intuitions of non-monotonic reasoning [2]. Moreover some of these variants are equivalent to other non-monotonic formalisms [3].

- DL integrates the concept of priorities between rules, thereby supporting a direct way of modeling preferences, without having to attach a metrics to them, as in the case of approaches based on utility functions [28].

- DL has a linear complexity, and existing implementations are able to deal with non trivial theories consisting of over 100,000 rules [25], offering thus an executable and scalable system.

A defeasible theory, i.e., a knowledge base in defeasible logic, consists of six different kinds of knowledge: facts, strict rules, defeasible rules, defeaters, a superiority relation $\succ$, and a specification of conflicting literals.

Facts denote simple pieces of information that are deemed to be true regardless of other knowledge items. A typical fact is that Tweety is an emu: $emu(Tweety)$.

Strict and defeasible rules are represented, respectively, by expressions of the form $A_1, \ldots, A_n \rightarrow B$ and $A_1, \ldots, A_n \Rightarrow B$, where $A_1, \ldots, A_n$ is a possibly empty set of prerequisites and B is the conclusion of the rule.

Strict rules are rules in the classical sense: whenever the premises of a rule are given, we are allowed to apply the rule and get a conclusion. When the premises are indisputable (e.g. facts) then so is the conclusion. An example of a strict rule is "every emu is a bird". Written formally:

$$emu(X) \rightarrow bird(X).$$

Defeasible rules are rules that can be defeated by contrary evidence. An example of such a rule is "birds usually fly"; written formally:

$$bird(X) \Rightarrow fly(X).$$

The idea is that if we know that something is a bird, then we may conclude that it has the capacity of flying, *unless there is other evidence suggesting that it has not.*

Defeaters are a special kind of rules. They are used to prevent conclusions, but never to support them. For example

$$heavy(X) \rightsquigarrow \neg fly(X)$$

This rule states that we have some reasons to believe that heavy things may not fly; however our belief is a weak one and we do not want to conclude that something does not fly only because it is heavy.

The *superiority relation* among rules is used to define priorities among rules, that is, where one rule may override the conclusion of another rule. For example, given the defeasible rules

$$\begin{array}{lll} r: & bird(X) & \Rightarrow fly(X) \\ r': & emu(X) & \Rightarrow \neg fly(X) \end{array}$$

which contradict one another, no conclusive decision can be made about whether an emu can fly. But if we introduce a superiority relation $\succ$ with $r' \succ r$, then we can indeed conclude that the emu does not fly.

In this paper, we assume that the superiority relation is acyclic.

For each literal p we define the set of *p-Complementary literals* $\mathcal{C}(p)$, that is, the set of literals that cannot hold when p does. Let us consider an example: suppose we have the

predicates *married* and *bachelor*. Here, we define, for any constant a, $\mathcal{C}(married(a)) = \{\neg married(a), bachelor(a)\}$. We know that, under the usual interpretation of the predicates they cannot be true at the same time for one and the same individual. We stipulate that the negation of a literal is always complementary to the literal.

We now give a short informal presentation of how conclusions are drawn in Defeasible Logic. A conclusion P can be derived if there is a rule whose conclusion is P, whose prerequisites (antecedent) are either already been proved or given in the case at hand (i.e. facts), and any stronger rule whose conclusion is $\neg P$ has prerequisites that fail to be derived. In other words, a conclusion P is derivable when:

- P is a fact; or
- there is an applicable strict or defeasible rule for P, and either
  - all the rules for $\neg P$ are discarded or
  - every rule for $\neg P$ is weaker than an applicable strict or defeasible rule for P.

For a formal definition of derivability in defeasible logic and a discussion on some of its properties see for example [5]. In what follows, we rather focus on the application of defeasible logic to automated negotiation. Specifically, we sketch a set of guidelines that can be used to formalize a negotiation strategy in defeasible logic. First of all, the negotiating agent developer needs to conduct an information analysis in order to identify an appropriate set of predicates to encode the negotiation strategy. These predicates must collectively capture the information characterising a given negotiation situation (e.g. negotiation issues, user parameters, thresholds, limit values, histories of offers, etc.) as well as the conclusions that can be derived from a negotiation situation (e.g. the actions that may need to be undertaken at a given point during the negotiation).

Next, the developer needs to identify the constraints over the negotiation, and the business rules governing the negotiation strategy. These constraints and business rules must then be classified as either hard or soft. Hard constraints (or hard business rules) are those that apply in any situation, regardless of the context. These constraints set the basic boundaries of the negotiating agent behaviour, and are used (for example) to model decisions that must be systematically taken when a given condition holds. Hard constraints and hard business rules are formalised directly as strict rules.

Soft constraints (or soft business rules) on the other hand can be violated under particular circumstances (i.e. they have exceptions). They are used to model the guidelines and user preferences that the negotiating agent will consider after making sure that all the hard constraints and rules are met. A soft constraint is formalized as a team of defeasible rules. Specifically the soft constraint itself is first translated into a defeasible rule in this team, and each exception to the constraint is then encoded as a separate defeasible rule, whose conclusion is the negation of the conclusion of the defeasible rule corresponding to the soft constraint. For example, if a soft constraint C is formalised by the defeasible rule r1: $A_1, \ldots, A_n \Rightarrow B$ (meaning that "normally" if $A_1, \ldots, A_n$ hold, then so should B),

and if we know that the state of affairs D constitutes an exception to C, then D must be formalized as r2: $D \Rightarrow \neg B$. Since D is an exception to C, we have to specify that r2 has precedence over r1, i.e. r2 $\succ$ r1.

In the third step, the developer should identify pairs of *incompatible litterals*. Two litterals are said to be incompatible if they cannot both hold at the same time, which essentially means that one of the litterals implies the negation of the other. Having identified conflicting litterals, and with the aid of an inference tool, the developer can then detect *conflicting (defeasible) rules*, i.e. rules such that the litterals appearing in the conclusions are incompatible. As we have alluded to above, no conclusion can be drawn from conflicting rules in defeasible logic, unless these rules are prioritised. For each pair of conflicting defeasible rules, the developer must analyse what will happen when the conjunction of the antecedents of the rules holds, and must deduce a priority between these rules from this analysis. In some cases, this analysis can be partially supported by an automated tool. Indeed, several criteria for automatically determining priorities among rules have been put forth, one of the most common being the *specificity* criterion. In a nutshell, a rule is more specific than another one, when it applies in all cases where the other does. In other words, the set of prerequisites of the more general rule is a subset of the set of prerequisites of the more specific rule. In such cases, it can be suggested to the developer to give a higher priority to the more specific rule. However, the developer may decide to do the opposite, since specificity alone is not always an appropriate criterion for ranking rules. In addition, in general, two rules cannot be compared according to the specificity criterion, and thus, the developer must either determine different ranking criteria or perform a manual ranking.

2.5 Glueing the control and the reasoning modules

In the example presented in figure 2, we used an informal syntax for describing the conditions and actions labeling the transitions and states of the statechart. It is actually in these conditions and actions, that the interaction between the control module and the other modules is expressed.

Specifically, the conditions labeling the arcs of the state-charts are queries to the knowledge base, while the actions fall within one of the following categories:

- Updates to non-derived predicates stored in the knowledge base.
- Activation of the reasoning module: that is, requesting the reasoning module to compute the value for one or several derived predicates and refresh the knowledge base accordingly.
- Sending a message to either a negotiating party or to the user through the communication module.

In the general case, these actions must be implemented using an external programming language. However, many of them can be automatically generated by a tool for building

negotiating agents. For instance, for each non-derivable predicate "P" appearing in the defeasible program, the tool could generate the corresponding action "set_P" (e.g. set_quote, set_current_bid, etc.). In addition, the tool could provide implementations for actions such as "revise(P)", which would compute a value of the derived predicate P, and update the knowledge base accordingly.

3 Case studies

Our approach to validate our proposal has been to use it in a number of case-studies involving a variety of negotiation situations. We have in particular applied our approach to design strategies for both bilateral and multi-party negotiation, and even to design agents capable of concurrently participating in several alternative negotiations. In this section, we detail two of the most representative case-studies that we have addressed: one concerning an English auction, and the other one concerning a one-to-many bargaining scenario, where one buyer simultaneously bargains with several potential sellers.

3.1 An English auction bidding agent

The *English auction* is perhaps one of the most popular one-to-many negotiation mechanisms. In its simplest form, it serves to select a buyer for an item and to establish its price (multi-party single-issue negotiation). There are many variants of the English auction (see [23] for a survey). The variant that is currently in use within the biggest online auction houses (e.g. *eBay* [10]) may be roughly described as follows. The seller starts by setting a *reservation price*, which may or may not be announced to the bidders. He also sets a *timing constraint*, which may be either expressed as a firm deadline, as a maximum duration between two successive bids, or as both. Potential buyers then issue increasingly higher bids. The *increment* between one bid and the next is constrained to be greater than a given threshold. The auction stops when the timing constraint is violated, i.e. either the deadline is reached, or no bid is registered for longer than the established maximum duration. The last bidder then buys the item at the price of the last bid. If no bid is issued at or above the reservation price, the item is not sold.

To illustrate how a bidder's strategy is expressed using defeasible logic, we consider the following scenario. A user wishes to participate in the auction of an item. (S)he doesn't know exactly how much the item is worth, but (s)he thinks that its value lies somewhere within two bounds L and U. The user is keen not to over-value the item, so (s)he decides to assume at the beginning of the auction that the item is worth L, and to eventually increase his valuation whenever one of the following two situations occurs: (a) at least 3 bids above his/her current valuation have been registered, or (b) somebody has bid more than 20% of his/her current valuation. As soon as one of these conditions is met, the user will raise his valuation by the minimum possible amount that allows him/her to stay in the auction. However, (s)he will never value the item above U. As usual in the case of English auctions, the user will start by bidding the reservation price (or the minimum

possible bid if the reservation price is not known), and (s)he will subsequently overbid the other participants' bids by the minimum increment, as long as the resulting bid is less than his/her current valuation. However, if the auction's deadline is too close, (s)he will bid his current valuation instead of just overbidding by the minimum increment[1].

Formally, the parameters, status, and history of the auction, are modeled through the following predicates and constants:

- Constant *min_increment* denotes the minimum amount by which the bidders are allowed to overbid.
- Constant *initial_bid* denotes the minimum amount of the first acceptable bid.
- Predicate $time\_remaining(T)$ gives the time remaining before the end of the auction.
- Predicate $highest\_quote(N)$ provides the current highest bid.
- Predicate $quotes\_above(X, N)$ holds if N bids above amount X have been registered.

The last two predicates (*highest_quote* and *quotes_above*) provide aggregate views over the history of the negotiation process as seen by the agent. This history is captured by a predicate $history(L)$, where L is a list of pairs $\langle time, proposal \rangle$. In the case of an auction, the proposals are the price quotes received by the agent from the auction broker. The rules for deriving predicates *highest_quotes* and *quotes_above* out of predicate *price_quotes* are all strict (i.e. not defeasible), and therefore we omit them in the sequel.

The parameters and decision rules of the user's strategy are modeled by the following constants and predicates:

- Constant *time_threshold* is the duration to the deadline, below which the user estimates that he should bid his/her valuation instead of just overbidding by the minimum increment.
- Constant $significant\_bidders$ is the number of bidders that should bid above The user's current valuation before (s)he considers raising it.
- Constant $significant\_increment$ is the amount (expressed as a percentage), that another bidder should bid above the user's current valuation before (s)he considers raising it (in the working example this is 0.2).
- Constant *max_valuation* is self-explanatory.
- Predicates $submit\_bid(X)$ states that a bid of amount X should be submitted.
- Predicate $valuation(X)$ gives the current valuation while $pre\_valuation(X)$ gives the valuation that was valid at the end of the previous activation of the reasoning module.

[1]The reasoning module that we develop here can be easily adapted to capture strategies where the increment between one bid and the next is gradually increased as the deadline approaches.

- Predicate $my\_bid(X)$ gives the amount of the last accepted bid issued by the bidder. At the beginning of the auction $my\_bid(0)$ holds.

The rules modeling the strategy are:

r1: $my\_bid(X), highest\_quote(Y), valuation(Z), X < Y, Y + min\_increment < Z,$
$time\_remaining(T), T > time\_threshold \Rightarrow submit\_bid(Y + min\_increment)$

r2: $my\_bid(X), highest\_quote(Y), valuation(Z), X < Y, Y + min\_increment < Z,$
$time\_remaining(T), T \leq time\_threshold \Rightarrow submit\_bid(Z)$

r3: $pre\_valuation(X) \Rightarrow valuation(X)$

r4: $pre\_valuation(X), quotes\_above(X, N), N \geq significant\_bidders, highest\_quote(Y)$
$\Rightarrow valuation(Y + min\_increment)$

r5: $pre\_valuation(X), highest\_quote(Y), Y > (1 + significant\_increment) \times X$
$\Rightarrow valuation(Y + min\_increment)$

r6: $Y > max\_valuation \rightsquigarrow \neg valuation(Y)$.

Rules r4 and r5 have precedence over r3.

The sets of complementary literals state that there can only be one amount to bid, and one new valuation, i.e.

- $\mathcal{C}(submit\_bid(x)) = \{\neg submit\_bid(y) \mid y \neq x\}$
- $\mathcal{C}(new\_valuation(x)) = \{\neg new\_valuation(y) \mid y \neq x\}$

Rules r1 and r2 model the bidding strategy: Rule r1 states that if there is enough time remaining and the agent's current bid is not the highest one, it should be increased by the minimum increment, provided that the current valuation allows so. Rule r2 states that if the deadline is close and the bidder does not hold the item, a bid of the amount of the current valuation should be submitted immediately. Rules r3 through r6 model the evolution of the valuation: Rule r4 and r5 model the two conditions under which the valuation should be raised, while rule r6 is a defeater modeling the fact that the bidder is under no circumstances willing to value the item above a given amount. The use of this defeater provides a strong modularity to the defeasible program. If for instance the user wanted to modify the above strategy with a statement of the form *"raise the valuation if the reservation price has not been met and the highest bid is above my current valuation"*, then (s)he just has to extend the above defeasible logic program with the following rule :

r7 $reservation\_not\_met, valuation(X), highest\_quote(Y), Y > X$
$\Rightarrow valuation(Y + min\_increment)$
r7 $\succ$ r3

without having to worry whether the reservation price is greater than his/her maximum valuation or not.

The control module of the bidding agent is described as a statechart in figure 3. The left region of the statechart is responsible for invoking the inference engine. At the beginning of the process, the knowledge base contains the fact $\partial\, submit\_bid(initial\_bid)$. Accordingly, the agent contacts the auctioneer to submit a bid (using action *send_bid*). If the bid is accepted, the agent receives a message which, when processed by the communication layer, generates the event *bid_accepted*. When a bid is accepted, the agent introduces a fact of the form $my\_bid(X)$ into the knowledge base and waits until a new quote is inserted. If on the other hand, a submitted bid is rejected, this means that another participant bid the same or a higher amount before the submission was processed, and therefore, the agent receives a quote with this amount instead of an acceptance message. Quotes are handled by the rightmost region of the statechart, which inserts them into the knowledge base, and generates a *quote_inserted* event after each insertion. These events are consumed by the left concurrent region.

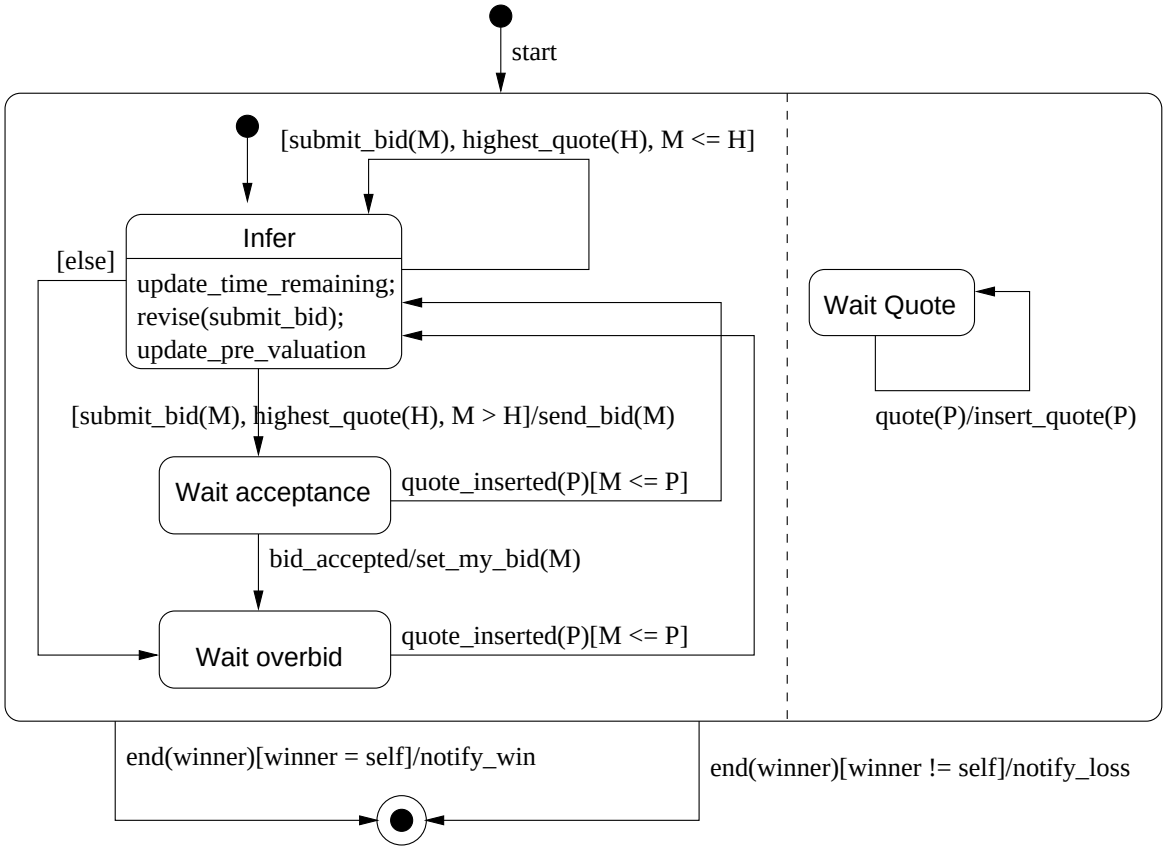

Figure 3: Specification of the control module of a bidding agent in an English auction.

When the inference engine cannot deduce a fact of the form $\partial\, submit\_bid(M)$, the control module takes the transition labeled "else" in figure 3, which leads to state "Wait

overbid". From then on, it waits for new quotes and when it receives one, it updates the knowledge base and invokes the inference engine again.

As stated in section 2.5, the conditions labeling the transitions of the statechart are queries to the knowledge base, while the actions correspond to updates, as well as invocations of the communication and the reasoning module. Some of these actions, such as "revise", "set_previous_valuation" and "set_my_bid" are predefined or can be automatically generated from the profiles of the corresponding predicates. Others such as "notify_win" and "notify_loss" have to be coded in an external programming language.

3.2 Bargaining with multiple parties

In a bargaining setting, the negotiation process begins with an initial offer formulated by one of the parties, and proceeds with an alternate exchange of offers, which stops when either an agreement is reached, or one of the parties decides to terminate the negotiation without an agreement. In many realistic scenarios, a party is not involved in a single bargaining process at a time, but rather, several bargainings over the same issue take place concurrently. This is the case for an individual who contacts several merchants in search of the best terms for buying a product. Under this perspective, the decision to terminate a given negotiation thread (i.e. a negotiation with a given party), and even the strategy for formulating a counter-offer, is dependent on the progress and outcomes of other negotiations threads. A negotiating agent for this kind of scenarios should therefore have a global view of all the negotiations in which the party that it represents is involved.

In this section, we focus on single-issue bargaining with non-binding offers. By "non-binding" we mean that when a party P1 sends an offer to another party P2, if P2 wishes to accept this offer, it needs to ask P1 for confirmation. It is not until P1 confirms its offer that that the negotiation is terminated with an agreement. If P1 refuses to confirm its offer, then it must either formulate a new offer or terminate the negotiation. This protocol is appropriate for situations where an agent negotiates with several others concurrently. Indeed, if offers were binding, an agent sending an offer would have to wait for the offer to be accepted or rejected before sending any offer to another party, and the concurrency would thus be limited.

We consider an agent responsible for negotiating a price with several potential sellers on behalf of a buyer. The control module of this agent is modeled as a statechart in figure 4. Offers sent by the sellers are arranged in a queue. The left concurrent region of the statechart describes how the offers are unqueued and evaluated. The outcome of the evaluation of an offer can be one of three actions: accept the offer, propose a counter-offer, or terminate the negotiation with the corresponding party. If an offer is accepted, the negotiating agent asks the other party to confirm the agreement, and upon receiving confirmation, it terminates the whole negotiation process (i.e. all the threads). If on the other hand a message of confirmation refusal is received, or the confirmation message is not replied after an agreed delay, an occurrence of event "no_confirmation(P,F)" is generated, and the agent continues to negotiate with the other parties.

The rightmost region of the statechart is responsible for receiving offers and termina-

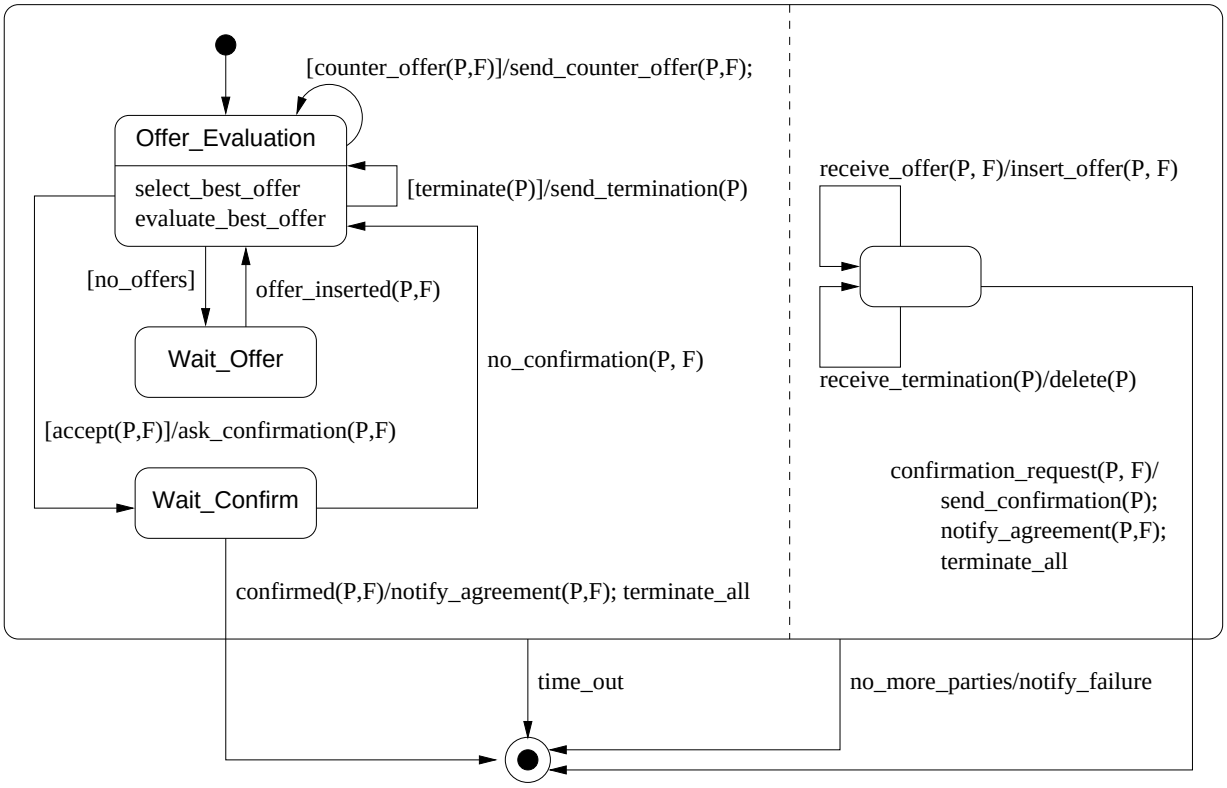

Figure 4: Specification of the control module of a multi-party bargaining agent.

tion messages from the sellers. When there are no more potential sellers remaining, the negotiation process is terminated with a failure. Similarly, when a party is willing to accept the last counter-offer that has been sent to it (i.e. a confirmation for a counter-offer is requested), a confirmation message is sent and the whole negotiation process is terminated.

Notice that it is not the buyer but the sellers who are responsible for generating the initial offers. Accordingly, we will use the term "offer" to designate the prices proposed by the sellers, and the term "counter-offer" to refer to the prices proposed by the buyer.

The reasoning module of the negotiating agent is composed of two separate defeasible logic programs: one for selecting the next offer to be evaluated, and the other for evaluating an offer. The rules for selecting an offer are simple (pick the offer with the lowest price among those which have not yet been evaluated), and we therefore omit their description.

The predicates for evaluating a given offer are:

- $\mathit{offer}(P, X)$: The current offer of party P is X (we assume that X is an integer).
- $\mathit{previous\_offer}(P, X)$: The previous offer of party P was X. If P has not made any previous offer, then the defeasible logic inference engine will deduce $\neg \mathit{previous\_offer}(P, X)$.
- $\mathit{counter\_offer}(P, X)$: The reasoning module suggests to reply to P's most recent offer with counter-offer X.

- *previous_counter_offer*(P, X) : The counter-offer suggested by the reasoning module in response of P's second most recent offer was X.
- *accept*(P) : The inference engine suggests to accept P's most recent offer.
- *terminate*(P) : The inference engine suggests to stop negotiating with P.

Finally, the rules involving these predicates are:

r1: *offer*$(P, Y), \neg$*previous_offer*$(P, Z), Y >$ *good_offer* $\Rightarrow$ *counter_offer*$(P,$ *good_offer*$)$
If this is the first offer from P and that it is not good enough, ask for a good price.

r2: *offer*(P, Y), *previous_offer*$(P, Z), Y \geq Z \Rightarrow$ *terminate*(Z)
If P has not improved his offer with respect to the previous one, stop negotiating.

r3: *offer*(P, Y), *previous_offer*$(P, Z), Y < Z$, *previous_counter_offer*(P, X), *best_offer*(W)
$\Rightarrow$ *counter_offer*$(P, min(X -$ *increment*$, W))$
If P has improved his offer with respect to the previous one, then concede a small amount with respect to the previous counter-offer. However, do not counter-propose a price below the current best offer.

r4: *few_parties*, *offer*(P, Y), *best_offer*$(Z), Y \leq Z, Y \leq$ *acceptable_price* $\Rightarrow$ *accept*(P, Y)
If there are few parties remaining and that the offer of P is acceptable and it is the the best one that has been received, accept it. A similar rule could be used to take into account the situation where the deadline for reaching an agreement is close.

r5: *offer*$(P, Y), Y \leq$ *good_price* $\Rightarrow$ *accept*(P, Y)
If somebody offers a good price, accept it.

Rules r4 and r5 both have precedence over r1, r2 and r3.

4 Related work

The Michigan AuctionBot [27, 38] is an auction management server supporting the creation, location and enactment of different kinds of auctions. Users can manually interact with the system through an HTML-based interface, or alternatively, they can develop their own arbitrarily complex bidding agents, and connect them to the auction manager through a TCP-level API. This API is generic enough to deal with several kinds of auctions (e.g. English, Dutch, double, etc.) through a common set of primitives. The AuctionBot has recently been used to host a trading agent competition [33], as has another similar platform called FishMarket [20]. Unlike our work, neither FishMarket nor the AuctionBot, address the issue of specifying bidding agent strategies.

Many efforts in the area of automated negotiation have focused on applying game theory techniques, either to design a protocol that encourages the participants to adopt a desirable strategy, or to design an optimal strategy for a given protocol [30, 32]. Although this

approach yields interesting results under simplifying assumptions (e.g. known valuations, risk-neutral attitudes, computationally unbounded agents), it is difficult or impossible to apply them in some realistic situations [22]. For this reason, [11] advocates the use of heuristic-based approaches in cases where game theoretic techniques are not applicable. Our work is complementary to the above ones, since we do not address the issue of designing strategies, but rather that of specifying them in an executable form.

[13] suggests to use a "variant" of defeasible logic to express strategies for agents trading over stock markets. The defeasible logic considered in this reference does not support an explicit ranking of the rules within a theory, but rather derives this ranking through a specificity criteria over arguments. Roughly speaking, an argument is more specific than (and therefore can defeat) another argument, if it takes into account more information. In addition to the fact that this approach allows tie-breaks between defeasible arguments, it may sometimes lead to counter-intuitive situations, since the semantics given by a user to the concept of "more information", may potentially not be in line with that given by the defeasible logic's inference method. Another important difference between the above proposal and ours, is that the control module is expressed through Prolog rules. We believe that event-driven formalisms such as statecharts, are more appropriate for this purpose.

Grosof et al. [8] use Courteous Logic Programming (CLP) to express knowledge about user preferences, constraints, and negotiation structures. The authors do not address the issue of specifying bidding strategies, but rather that of determining the set of auctions and other negotiations that need to be undertaken in order to transform a contract template into an executable contract. Interestingly, Defeasible Logic (DL) is more expressive than CLP, in the sense that it fully supports stratified theories [3]. Hence, the question remains open whether it is possible to express "useful" contract templates in DL that are not expressible in CLP.

In this paper, we did not address the issue of inter-agent message exchange. For some negotiation protocols such as the English and the Dutch auctions, some standardisation proposals are being developed by the FIPA organisation [12]. Also, inter-agent communication protocols such as KQML [34] could be used for this purpose.

5 Conclusion

We have proposed a pragmatic approach to negotiating agents development, which seamlessly combines an imperative formalism (statecharts) with a declarative one (defeasible logic). The choice of these two formalisms has been justified against a set of desirable criteria for negotiation strategy specification languages, and their suitability has been validated through concrete examples of non-trivial negotiation strategies. In particular, we have shown that this approach is applicable for developing agents capable of participating in multiple concurrent negotiations.

All the defeasible logic programs appearing in the paper have been tested using a defeasible logic inference engine [26]. Currently, the translation of the statechart into an executable program is carried out manually. However, it is clear that this process could be

automated, and as a matter of fact, there exist several commercial tools for generating code from a statechart (e.g. Statemate Magnum [21]). Alternatively, an interesting approach would be to develop an interpreter of statecharts, which combined with a defeasible logic inference engine, and an appropriate inter-agent communication library, could be used to build an agent shell capable of dynamically loading and executing negotiation strategies expressed in our framework.

In this paper, we have advocated the use of (defeasible) rules to specify the decision-making aspects of negotiation strategies. It is conceivable however to use rules not only for specifying negotiation strategies, but also for expressing the offers and counter-offers exchanged between agents during a negotiation. Allowing agents to exchange rules, instead of just exchanging simple communication performatives, significantly increases the flexibility of a negotiation protocol. Indeed, agents can then dynamically add, remove, or modify issues. They can also express entire spaces of acceptable deals in a single message, instead of expressing only one acceptable deal per message. This idea, which is suggested in [37, 8], is certainly worth further investigation.

Another avenue for future work is the integration of defeasible logic, as a formalism for qualitatively expressing preferences, with a quantitative formalism such as Multi-Attribute Utility Theory (MAUT). Indeed, the application of MAUT to capture preferences and trade-offs over multiple issues in the context of both manual and automated negotiations, has been thoroughly studied [28, 11, 4]. MAUT-based approaches to strategy specification are especially suitable in situations where it is intuitive to attach metrics to both the negotiation issues and the user preferences. In contrast, the logic-based approach presented in this paper is suitable when there is no intuitive way of defining such metrics, but instead the user has in mind a set of prioritised guidelines to find acceptable deals. The marriage of these two types of formalisms (quantitative and qualitative) can thus lead to a powerful approach for specifying negotiation strategies.

References

[1] G. Antoniou. *Nonmonotonic Reasoning*. MIT Press, 1997.

[2] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. A flexible framework for defeasible logics. In *Proc. American National Conference on Artificial Intelligence (AAAI)*, pages 405–410. AAAI/MIT Press, 2000.

[3] Grigoris Antoniou, Michael J. Maher, and David Billington. Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming*, 42(1):47–57, 2000.

[4] M. Barbuceanu and W.-K. Lo. A multi-attribute utility theoretic negotiation architecture for electronic commerce. In *Proceedings of the International Conference on Autonomous Agents*, pages 239–246, Barcelona, Spain, May 2000. ACM Press.

[5] D. Billington. Defeasible logic is stable. *Journal of Logic and Computation*, 3:370–400, 1993.

[6] Marco Cadoli and Marco Schaerf. A survey of complexity results for nonmonotonic logics. *Journal of Logic Programming*, 17:127–160, 1993.

[7] A. Chavez, D. Dreilinger, R. Guttman, and P. Maes. A real-life experiment in creating an agent marketplace. In *Proc. of the 2nd Int. Conference on The Practical Applications of Agents and Multi-Agent Technology (PAAM)*, London, UK, 1997.

[8] B. N. Grosof D. M. Reeves, M. P. Wellman. Automated negotiation from declarative contract descriptions. In *Proceedings of the International Conference on Autonomous Agents*, pages 51–58, Montreal, Canada, May 2001. ACM Press.

[9] M. Dumas, L. Aldred, G. Governatori, A. ter Hofstede, and N. Russell. Probabilistic automated bidding in alternative auctions. In *Proc. of the 11th International Conference on the World Wide Web (WWW)*, Honolulu HI, USA, May 2002. ACM Press.

[10] eBay. Home page. `http://www.ebay.com`.

[11] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3–4):159–182, 1998.

[12] Foundation for Intelligent Physical Agents. FIPA Interaction Protocols Specification Repository. `http://www.fipa.org/repository/ips.html`.

[13] A. Garcia, D. Gollapally, P. Tarau, and G. Simari. Deliberative stock market agents using Jinni and defeasible logic programming. In *Proc. of the ECAI Workshop on Engineering Societies in the Agents' World*, Berlin, Germany, August 2000. Springer Verlag.

[14] E. Giménez-Funes, L. Godo, J.A. Rodriguez-Aguilar, and P. García. Designing bidding strategies for trading agents in electronic auctions. In *Proc. of the 3rd Int. Conference on Multi-Agent Systems (ICMAS)*, Paris, France, July 1998.

[15] G. Governatori, A. ter Hofstede, and P. Oaks. Defeasible logics for automated negotiation. In *Proc. of the CollECTeR Conference on Electronic Commerce*, Brisbane, Australia, December 2000.

[16] Guido Governatori and Michael J. Maher. An argumentation-theoretic characterization of defeasible logic. In Werner Horn, editor, *ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence*, Amsterdam, 2000. IOS Press.

[17] J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base*. ISO/TC97/SC5/WG3-N695, ANSI, New York, USA, 1982.

[18] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.

[19] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[20] IIIA, ISOCO and UPC (organizers). AMECIII trading agents' tournament. `http://www.iiia.csic.es/Projects/fishmarket/agents2000`, June 2000.

[21] ILogix Inc. Statemate Magnum. `http://www.ilogix.com`.

[22] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *Journal of Group Decision and Negotiation*, 10(2), March 2001.

[23] D. Lucking-Reiley. Auctions on the Internet: What's being auctioned, and how? *Journal of Industrial Economics*, 48(3):227–252, September 2000.

[24] P. Maes, R. Guttman, and A. Moukas. Agents that buy and sell. *Communications of the ACM*, 42(3):81–91, March 1999.

[25] Michael J. Maher, Andrew Rock, Grigoris Antoniou, David Billington, and Tristan Miller. Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools*, 10(4):483–501, 2001.

[26] D. Nute et al. Defeasible logic implementation. Accessed from `ftp://ftp.ai.uga.edu/pub/prolog` on 14 March 2001.

[27] The University of Michigan Artificial Intelligence Laboratory. The Michigan Internet AuctionBot. `http://auction.eecs.umich.edu`.

[28] H. Raiffa. *The art and science of negotiation*. Harvard University Press, Cambridge, MA, 1982.

[29] Reticular Systems Inc. Agent construction tools. `http://www.agentbuilder.com/AgentTools`.

[30] J.S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Readings MA, USA, 1994.

[31] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[32] T. Sandholm. Distributed rational decision making. In Weiss [35].

[33] Betsy Strother. TAC: A Trading Agent Competition. *ACM SIGeCom Exchanges*, 1(1), August 2000.

[34] University of Maryland Baltimore County. KQML. http://www.cs.umbc.edu/kqml.

[35] G. Weiss, editor. *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*. MIT Press, 1999.

[36] M. Wooldridge. Intelligent agents. In Weiss [35].

[37] M. J. Wooldridge and S Parsons. On the use of logic in negotiation. In *Proceedings of the Workshop on Agent Communication Languages*, Barcelona, Spain, May 2000. Accessed from http://www.csc.liv.ac.uk/~sp/publications/conferences/acl00.html on 18 July 2002.

[38] P.R. Wurman, M.P. Wellman, and W.E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proc. of the 2nd Int. Conference on Autonomous Agents*. ACM Press, May 1998.